```xml
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
</dependencies>
```

view raw spring-boot-dto-tutorial.xml hosted with ❤ by **GitHub**

The DTO

```java
public class PersonDTO {

  private String firstName;
  private String secondName;
  // Formats output date when this DTO is passed through JSON
  @JsonFormat(pattern = "dd/MM/yyyy")
  // Allows dd/MM/yyyy date to be passed into GET request in JSON
  @DateTimeFormat(pattern = "dd/MM/yyyy")
  private Date dateOfBirth;

  private String profession;
  private BigDecimal salary;

  public PersonDTO(
      String firstName, String secondName, Date dateOfBirth, String profession, BigDecimal salary) {
    this.firstName = firstName;
    this.secondName = secondName;
    this.dateOfBirth = dateOfBirth;
    this.profession = profession;
    this.salary = salary;
  }

  public PersonDTO() {}

  public String getFirstName() {
    return firstName;
  }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  public String getSecondName() {
    return secondName;
  }

  public void setSecondName(String secondName) {
    this.secondName = secondName;
  }

  public Date getDateOfBirth() {
    return dateOfBirth;
  }

  public void setDateOfBirth(Date dateOfBirth) {
    this.dateOfBirth = dateOfBirth;
  }

  public String getProfession() {
    return profession;
  }

  public void setProfession(String profession) {
    this.profession = profession;
  }

  public BigDecimal getSalary() {
    return salary;
  }

  public void setSalary(BigDecimal salary) {
    this.salary = salary;
  }
}
```

view raw PersonDTO.java hosted with ❤ by **GitHub**

A few things to notice

```
@JsonFormat(pattern = "dd/MM/yyyy")
```

Formats the date when the DTO is output to JSON. If this is not used the JSON will display a number that represents the time instead of a easy to read string.

```
@DateTimeFormat(pattern = "dd/MM/yyyy")
```

This works the other way around as it allows the date to be input in *dd/MM/yyyy* format, which if your trying to pass a date directly into JSON it will be hard to know the number version of the date you want.

Another important thing to notice is that there are setters for all the fields in the DTO. Without these fields the values passed into JSON will not be set onto the DTO and will be left as null.

The controller

```
@RestController
public class PersonRestController {

  private static final SimpleDateFormat simpleDateFormat = new SimpleDateFormat("dd/MM/yyyy");

  @RequestMapping("/getWithRequestParam")
  public List<Object> getWithRequestParam(@RequestParam(value = "personDTO") String personDTO)
      throws IOException {
    final PersonDTO person =
        new ObjectMapper().setDateFormat(simpleDateFormat).readValue(personDTO, PersonDTO.class);
    return Arrays.asList(
        person.getFirstName(),
        person.getSecondName(),
        person.getDateOfBirth(),
        person.getProfession(),
        person.getSalary());
  }

  @RequestMapping("/getWithoutRequestParam")
  public List<Object> getWithoutRequestParam(PersonDTO personDTO) {
    return Arrays.asList(
        personDTO.getFirstName(),
        personDTO.getSecondName(),
        personDTO.getDateOfBirth(),
        personDTO.getProfession(),
        personDTO.getSalary());
  }

  @RequestMapping(value = "/getWithMultipleParameters")
  public List<Object> getWithMultipleParameters(
      PersonDTO personDTO, @RequestParam(value = "firstName") String firstName) {
    return Arrays.asList(
        personDTO.getFirstName(),
        personDTO.getSecondName(),
            personDTO.getDateOfBirth(),
            personDTO.getProfession(),
            personDTO.getSalary(),
            firstName);
    }

    @RequestMapping("/getWithMultipleRequestParams")
    public List<Object> getWithMultipleRequestParams(
        @RequestParam(value = "personDTO") String personDTO,
        @RequestParam(value = "firstName") String firstName)
        throws IOException {
      final PersonDTO person =
            new ObjectMapper().setDateFormat(simpleDateFormat).readValue(personDTO, PersonDTO.class);
      return Arrays.asList(
            person.getFirstName(),
            person.getSecondName(),
            person.getDateOfBirth(),
            person.getProfession(),
            person.getSalary(),
            firstName);
    }
}
```

view raw PersonRestController.java hosted with ❤ by **GitHub**

This example shows various ways of passing the DTO to a *RestController* which in this simple example extracts the values from the DTO and returns them in a *List<Object>*.

All of the handlers are marked with the annotation *@RequestMapping* which specifies that it is a handler and can accept requests.

```
@RequestMapping("/getWithRequestParam")
```

This line will map requests from the path (mine is using localhost)

```
localhost:8080/getWithRequestParam
```

By default this mapping will accept GET requests. A different way to write this annotation if you want to clearly state that it accepts GET requests is

```
@RequestMapping(value = "/getWithRequestParam", method = RequestMethod.GET)
```

Now lets look at each handler more closely.

```
@RequestMapping("/getWithRequestParam")
public List<Object> getWithRequestParam(@RequestParam(value = "personDTO") String personDTO)
    throws IOException {
  final PersonDTO person =
      new ObjectMapper().setDateFormat(simpleDateFormat).readValue(personDTO, PersonDTO.class);
  return Arrays.asList(
      person.getFirstName(),
      person.getSecondName(),
      person.getDateOfBirth(),
      person.getProfession(),
      person.getSalary());
}
```

view raw GetWithRequestParam.java hosted with ❤ by **GitHub**

In this piece of code *@RequestParam* is used which will give a name to a parameter that needs to be passed into it via the request. I personally used Postman to send these requests and test this code.

```
localhost:8080/getWithRequestParam?personDTO=
{"firstName":"Dan","secondName":"Newton","profession":"Java
Developer","salary":1234,"dateOfBirth":"06/01/1994"}
```

This is the input the handler takes in. The *personDTO* parameter is specified after the *?* as its values are to be passed to the handler. The curly brackets specify the start and end of the values that make up the *personDTO* object.

As the *personDTO* is passed in as a string it needs to be mapped to an actual *PersonDTO* object before anything can be done with it. This is done by using the *ObjectMapper* which takes in JSON string and outputs a specified object using the parsed values from the string. Also notice that I used *setDataFormat* as in my request I passed the date in the *dd/MM/yyyy* format which is not the default that it will attempt to parse and therefore it will fail without it.

This produces the JSON output

```
[
 "Dan",
 "Newton",
 757814400000,
 "Java Developer",
 1234
]
```

```
@RequestMapping("/getWithoutRequestParam")
public List<Object> getWithoutRequestParam(PersonDTO personDTO) {
  return Arrays.asList(
          personDTO.getFirstName(),
                personDTO.getSecondName(),
                personDTO.getDateOfBirth(),
                personDTO.getProfession(),
                personDTO.getSalary());
}
```

view raw GetWithoutRequestParam.java hosted with ❤ by **GitHub**

The key difference between this example and the previous is that there is no *@RequestParam* annotation and a *PersonDTO* is passed directly into the handler. As the object is passed in directly there is nothing else to do at this point. The input this example takes is

```
localhost:8080/getWithoutRequestParam?firstName=Dan&secondName=Newton&profession=Java
Developer&salary=1234&dateOfBirth=06/01/1994
```

Notice that there is no mention of the *personDTO* anywhere and all the values passed in will be used to set the properties of the DTO.

Now what happens when we start passing in the DTO with other objects?

```
@RequestMapping(value = "/getWithMultipleParameters")
 public List<Object> getWithMultipleParameters(
     PersonDTO personDTO, @RequestParam(value = "firstName") String firstName) {
   return Arrays.asList(
       personDTO.getFirstName(),
       personDTO.getSecondName(),
       personDTO.getDateOfBirth(),
       personDTO.getProfession(),
       personDTO.getSalary(),
       firstName);
 }
```

view raw GetWithMultipleParameters.java hosted with ❤ by **GitHub**

So here we are passing in the *PersonDTO* straight into the handler along with another request parameter. One important thing to notice is that the extra parameter has the same name as one of the properties of the *PersonDTO* object. Therefore setting the *firstName* parameter will set the value on the *personDTO* as well as the separate input marked on the handler.

This is also a good time to show you something else that is interesting. If we set a parameter twice in the URL it will append the original value with a comma plus the second instance's value.

```
localhost:8080/getWithMultipleParameters?firstName=Dan&secondName=Newton&profession=Java
Developer&salary=1234&dateOfBirth=06/01/1994&firstName=Another name
```

Leading to the JSON output of

```
[
 "Dan,Another name",
 "Newton",
 757814400000,
 "Java Developer",
 1234,
 "Dan,Another name"
]
```

Now this isn't really a desirable output, so is there a way to get around this? If we go back and use the *@RequestParam* annotation that we used earlier to pass in the *personDTO* where we need to define its values with a map of values and assign it tothe *personDTO* parameter in the request, we can get around this problem.

```
@RequestMapping("/getWithMultipleRequestParams")
public List<Object> getWithMultipleRequestParams(
  @RequestParam(value = "personDTO") String personDTO,
  @RequestParam(value = "firstName") String firstName)
  throws IOException {
  final PersonDTO person =
      new ObjectMapper().setDateFormat(simpleDateFormat).readValue(personDTO, PersonDTO.class);
  return Arrays.asList(
      person.getFirstName(),
      person.getSecondName(),
      person.getDateOfBirth(),
      person.getProfession(),
      person.getSalary(),
      firstName);
}
```

view raw GetWithMultipleRequestParams.java hosted with ❤ by **GitHub**

Which takes in the request

```
localhost:8080/getWithMultipleRequestParams?personDTO=
{"firstName":"Dan","secondName":"Newton","profession":"Java
Developer","salary":1234,"dateOfBirth":"06/01/1994"}&firstName=Another Name
```

And outputs the following JSON

```
[
 "Dan",
 "Newton",
 757814400000,
 "Java Developer",
 1234,
 "Another Name"
]
```

As you can see even though there are 2 *firstName* parameters in the request there is no appending of their values. As one consists as a value of the *personDTO* parameter whereas the other is a separate parameter.

In this tutorial we investigated how to pass a data transfer object with a GET request to a Spring Rest Controller. The code used in this post can be found on my Github.